Modelling and Verification of Business Processes*

Costin Bădică
Department of Computer Science
King's College
London, WC2R 2LS, UK
badica@dcs.kcl.ac.uk

Chris Fox
Department of Computer Science
University of Essex
Colchester, CO4 3SQ, UK
foxcj@essex.ac.uk

ABSTRACT
This paper introduces a notation for business process modelling and shows how it can be used for static verification of business processes under the assumption of single instance executions. Our approach is based on formalizing business process models as flownomial expressions [4]) and evaluating them over boolean relations. Its main advantage is simplicity: it is based on evaluating algebraic expressions to boolean relations. But it is also more restrictive then other approaches because, basically, it can only indicate those input patterns that provided to a process can cause it to enter an infinite loop without escape or a resource starvation. Nevertheless, this is useful within a tool to check processes, in order to point out problems as early as possible, before running any dynamic simulation.

KEY WORDS
Business Process Modelling, Verification, Flownomial Calculus

1 Introduction

An increased interest in applying information technology to the field of business processes has been manifested during the last decade, both in research and commercial communities, as a response to slogans like business process re-engineering or total quality management. This is proven by the large number of papers published on the subject, the large number of emerging standards and proposals for representing different aspects of business processes and the large number of software tools that support tasks like business process modelling, design, analysis or simulation.

Informally, by a *business process* we mean a process that is carried out in an organization in order to achieve the organization business objectives. Because organizations are very complex artifacts. it has been claimed that carefully developed models are necessary for describing, analyzing and/or enacting the underlying business processes ([1]). A business process model is usually expressed by means of a graphical notation, that must be able to capture all the relevant information from the model and additionally must facilitate the analysis of the modelled process by static verification and/or dynamic simulation.

Analysis of business process models is very important in the context of *business process re-engineering* (BPR hereafter), because the task of BPR is to evaluate the current processes with the goal of radically revising them, in order to accommodate their improvement to new organizational needs or goals.

The INSPIRE project (IST-10387-1999) aims to develop an integrated tool-set to support a systematic and more human-oriented approach to BPR. A central aspect in INSPIRE was the development of a notation for representing business process models that is both easy to use and understand by the project partners (consultants and developers; note that consultants are not usually IT experts) and also sufficiently rigorous to allow static verification and dynamic simulation – essential tasks within a BPR context ([11]).

The notation of business process models employed in INSPIRE combines features of IDEF0 ([2]) and IDEF3 ([3]). We started with an IDEF0-based notation and then enhanced it with facilities for representing the dynamics of a business process, based on the process schematics employed in IDEF3 ([3]).

This paper introduces the INSPIRE notation and shows how it can be used for static verification of business processes under the assumption of single instance executions. Our technique is based on formalizing business process models as flownomial expressions [4]) and evaluating them over boolean relations. Other approaches proposed in the literature for verifying business processes are: model checking ([5]), process algebras [6]), knowledge-based systems ([7]) or Petri nets ([8]). One advantage of our approach is simplicity: it is based on evaluating algebraic expressions to boolean relations. Our approach is also more restrictive then the ones above. Basically, it can only indicate those input patterns that provided to a process can cause it to enter an infinite loop without escape or a resource starvation. Nevertheless, this is useful in INSPIRE to check processes and to point out problems as early as possible, before running any dynamic simulation.

The paper is structured as follows: section 2 gives

*The work described here was supported by funding from the European Union as part of the Framework V INSPIRE project (IST-10387-1999)

an informal introduction of the notation and the rationale behind it; section 3 shows how we can describe our models as flownomial expressions; section 4 shows how we can verify a business process by evaluating flownomial expressions; section 5 concludes the paper.

2 A Notation For Business Processes

2.1 The Basic Black Box Model

IDEF0 is a technique used to produce a function model of a new or existing system or subject area. The result of applying the IDEF0 technique is an IDEF0 model of the system. An important assumption stated in the IDEF0 standard ([2]) is: only that which is explicitly stated is necessarily implied. As a corollary, what is not (explicitly) prohibited is (implicitly) allowed. This shows that starting from an IDEF0-based notation and extending it in a consistent way is a correct approach. In our case, the extension is based on IDEF3 ([3]).

The modelling elements of IDEF0 are (i) *boxes* and (ii) *arrows*. *Boxes* represent functions defined as activities, processes or transformations and *arrows* represent data or objects related to functions. A *box* describes what happens in a designated function. A *box* has a set of inputs $\{i_1, \ldots, i_m\}$ and a set of outputs $\{o_1, \ldots, o_n\}$. We consider controls as a special kind of inputs. Mechanisms are not considered in our approach, although the notation employed by the INSPIRE tool supports them.

The crucial thing is, how we are to interpret these boxes? The IDEF0 standard ([2]) is very vague with respect to this. Basically it says that in order to produce any subset of the outputs $o_1, \ldots, o_n$ any subset of the entries $i_1, \ldots, i_m$ may be required. In order to understand this statement, we must first recognize that one intuitive interpretation in mathematical terms of an IDEF0 box is a function taking m inputs and producing n outputs. This function actually describes how we can compute the outputs on the basis on the given inputs. However, the "how" component is not explicitly modelled. Eventually, it is just suggested by the verb phrase that names the box.

Let us now return to the basic interpretation as stated in the IDEF0 standard ([2]). If our boxes are always modelling functions, it means that all the outputs will be produced if all the inputs are present. This turns out to be a very restrictive assumption, especially if the IDEF0 method is used in the early stage of requirements capture and specification. Obviously, we would not like to be very restrictive during this stage. Moreover, we would like to be able to model situations when even not all the inputs are present, at least some outputs will be produced. If we abstract away from the actual domains of values for the inputs and outputs and from how the outputs are actually produced, we finally arrive at an interpretation of boxes as boolean relations taking m inputs and producing n outputs. So formally, a box should be interpreted as an unspecified relation $rel \subseteq \{0,1\}^m \times \{0,1\}^n$.

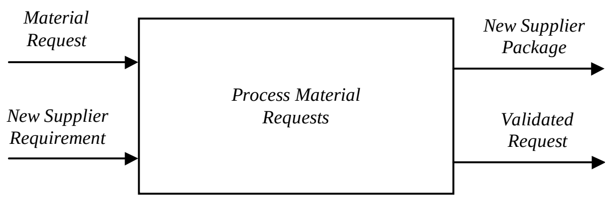

Figure 1. A business process for processing material requests

So, our boxes are in fact black-boxes describing the dependency of subsets of outputs on subsets of inputs. A dependency is a pair of boolean strings that just says that it is possible for the underlying activity modelled by the box to produce the specified outputs when the specified inputs are available. For example, the pair (011,100) says that it is possible to produce only output 1 if only inputs 2 and 3 are present.

As for the arrows, they should be interpreted as flows transporting data or object items from producers to consumers. However, special attention must be paid to branching arrows (forks and joins).

According to [2], a fork is an arrow from source to use that is divided into two or more arrows. Because the IDEF0 standard is too vague here, we make the following assumption: a fork indicates the fact that the item placed onto the arrow from source to use will be made available to all of its destinations. In this case all the arrows will have the same label. If the fork is not intended to model this situation, than it must be replaced with a single input/multiple outputs activity.

Also, [2] states that a join is a point where two or more arrows from source to use are merged into a single arrow. Merging is different from splitting because it is difficult to imagine that it could happen outside an activity. That is why we have chosen to model joins as "dummy" multiple inputs/single output activities.

For example, in a manufacturing company we find a business process for material acquisition. It takes material requests and produces purchase orders and payment authorizations. It contains a subprocess for handling the material requests that takes material requests and produces validated requests. The company has a list of available suppliers, but is must be prepared to find and handle new potential suppliers. So, there is an additional input to handle new supplier requirements and an additional output to produce new supplier packages. The process is represented at an abstract level in figure 1. The IDEF0 technique allows the presentation of a model at different levels of detail. The process in figure 1 is detailed in figure 2. Figures

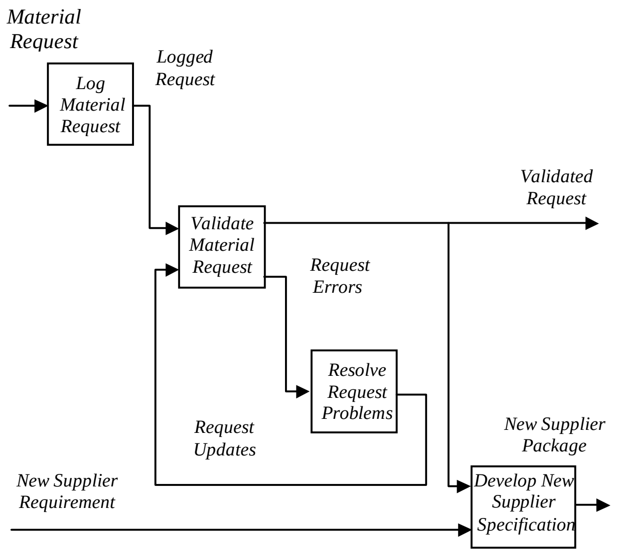

Figure 2. A more detailed presentation of the process for processing material requests

1 and 2 are also called *IDEF0 diagrams* ([2]).

2.2 Adding Glass-Box Views

Sometimes it is useful to be able to constrain the dependency of the outputs from the inputs. One way of doing this is by attaching to each black-box a glass-box view based on IDEF3. IDEF3 is a technique used to produce a dynamic model of the system. IDEF3 can produce two views of the system: a process-centered view and an object-centered view. Here we are considering only the process-centered view.

The main building blocks of the process-centered view of IDEF3 are (i) *units of behaviors*, (ii) *links* and (iii) *junctions*. *Units of behavior* represent types of happenings (events, acts, etc.), *links* represent temporal relations between units of behavior, and *junctions* are used to specify the logic of process branching. Within the INSPIRE we are using the term *connector* instead of *junction*, so we shall use this term hereafter.

A glass-box view contains a unit of behavior, a tree of input connectors and a tree of output connectors. There are one-to-one mappings between the inputs of a black-box and the leaves of its input tree and between the outputs of a black-box and the leaves of its output tree. The unit of behavior models the "instantiation" of the activity. The input tree models the logic of selecting the inputs participating in the activity, and the output tree models the logic of generating the outputs produced by the activity.

Let us consider the business process in figure 2. We can associate glass-box views to activities *Validate Material Request* and *Resolve Request Problems*. *Validate Material Request* may take a *Logged Request* or *Request Updates* and may produce either a *Validated Request* or *Request Errors*. *Develop New Supplier Specification* takes both a *Validated Request* and a *New Supplier Requirement* to be able to produce a *New Supplier Package*. This is modelled in figure 3.

2.3 Some Comments

The "boxes as mathematical functions" interpretation is not new. It is in fact the basic assumption employed by the formally founded description technique of business processes reported in [9]. Also our concept of "glass-box view" is very similar to the one used in [10]. The main goal in [9] is to document single runs of exemplary system behavior. This is quite similar to the hypothesis we use in our verification technique, namely to focus on single instance executions. Other similarities between our notation and the one described in [9] and [10] are: the use of black boxes to define process signatures and the use of glass boxes to constrain the set of behaviors of a black box.

There are however three important differences between our work and the one reported in [9] and [10]. First, we abstract away from the actual domains of the inputs and outputs and from how the outputs are produced based on the provided inputs. We are interested only what outputs are produced when some inputs are provided. Second, a basic assumption in [9] and [10] is that the process nets are acyclic, to avoid circular dependencies. In our notation we cope also with circular process nets. The feedback operator of flownomial expressions ([4]) is used to provide a semantics for such constructs. Third, the semantics of the notation from [9] is based on functions and their composition, while our approach is based on boolean relations.

3 Modelling Business Processes As Flownomial Expressions

3.1 A Definition of Black-Boxes

We are using flownomial expressions ([4]) to describe the syntax of networks of black-boxes.

Let us first introduce some notations. For any set S let S^* be the set of finite length strings with elements in S. For any $x, y \in S^*$ let $xy \in S^*$ be the concatenation of x with y. The length of $x \in S^*$ is denoted by $|x|$, and let x_i to be the i-th component of x, for all $i \in \{1, \ldots, |x|\}$.

Let F be a set of *flow type names* and A be a set of *activity names*. The arrows and black-boxes in our model are labelled with flow type names and activity names, respectively. Additionally, any two distinct activities have distinct labels, and arrows labelled with

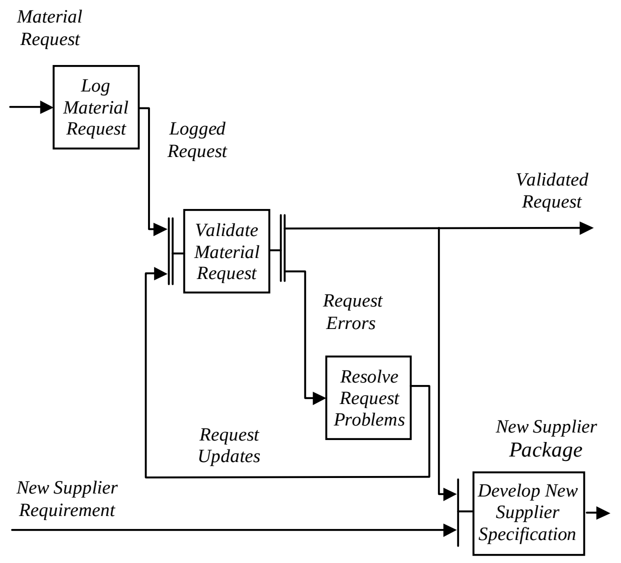

Figure 3. The result of adding glass-box views to the processing material requests process

the same flow type name represent flows carrying data of the same type.

To each activity label we associate an *arity* $r : A \to F^* \times F^*$. It specifies the type names of the flows attached to the inputs and the outputs of the activity. The triple (A, F, r) is called a *signature*.

We can define the following operations on elements in $F^* \times F^*$: i) if $r_i = (f_i, g_i) \in F^* \times F^*$ and $i = \{1, 2\}$, then $r_1 r_2 = (f_1 f_2, g_1 g_2)$; ii) if $r_i = (f_i, g_i) \in F^* \times F^*$, $i = \{1, 2\}$ and $g_1 = f_2$, then $r_1 \circ r_2 = (f_1, g_2)$; iii) if $r = (fh, gh) \in F^* \times F^*$ then $r \uparrow^h = (f, g)$. They are needed to model the propagation of arities when we compose complex networks from simpler elements.

An IDEF0 diagram can be described as a *flownomial expression* composed of constants, variables and operations. Let $\mathcal{E}(A, F, r)$ be the set of flownomial expressions over a signature (A, F, r). We can extend the arity function r over $\mathcal{E}(A, F, r)$. Flownomial expressions and their arities are defined as follows: *variables*: $a \in A$ is a flownomial expression of arity $r(a)$; ii) *constants*: $\mathbf{I}_f, f \in F^*$ is a flownomial expression of arity $r(\mathbf{I}_f) = (f, f)$; $\wedge_2^f, f \in F^*$ is a flownomial expression of arity $r(\wedge_2^f) = (f, ff)$; ${}^{f_1}\times^{f_2}, f_1, f_2 \in F^*$ is a flownomial expression of arity $r({}^{f_1}\times^{f_2}) = (f_1 f_2, f_2 f_1)$; iii) *operations*: if E_i are flownomial expressions, then $E_1 E_2$ is a flownomial expression of arity $r(E_1) r(E_2)$ called the *parallel composition* of E_1 and E_2; if E_1 and E_2 are flownomial expressions and $r(E_1) \circ r(E_2)$ is well-defined then $E_1 \circ E_2$ is a flownomial expression of arity $r(E_1) \circ r(E_2)$ called *sequential composition* of E_1 and E_2; if E is a flownomial expression of arity (fh, gh) then $E \uparrow^h$ is a flownomial expression of arity $r(E) \uparrow^h$; $\uparrow^h$ is called the *feedback operation.*

For example consider the process in figure 2. To avoid the use of long names, we rename the flows and activities as follows: *Material Request* $= i_1$, *New Supplier Requirement* $= i_2$, *Validated Request* $= o_1$, *New Supplier Package* $= o_2$, *Logged Request* $= t_1$, *Request Errors* $= t_2$, *Request Updates* $= t_3$, *Log Material Request* $= a_1$, *Validate Material Request* $= a_2$, *Resolve Request Problems* $= a_3$ and *Develop New Supplier Spec* $= a_4$. A flownomial expression for it is $E = (((a_1 \circ ((a_2 \circ (\mathbf{I}_{o_1} a_3)) \uparrow^{t_3})) \circ \wedge_2^{o_1}) \mathbf{I}_{i_2}) \circ (\mathbf{I}_{o_1} a_4)$. The signatures of a_1, a_2, a_3, a_4 are: $r(a_1) = (i_1, t_1)$, $r(a_2) = (t_1 t_3, o_1 t_2)$, $r(a_3) = (t_2, t_3)$, $r(a_4) = (o_1 i_2, o_3)$. Applying the rules for computing the arities of flownomial expressions we obtain $r(E) = (i_1 i_2, o_1 o_2)$.

3.2 An Interpretation of Black-Boxes

Flownomial expressions are intended to model IDEF0 diagrams. An important claim made in subsection 2.1 is that black-boxes are naturally interpreted as boolean relations. This interpretation can be extended to flownomial expressions, as well. To formalize this idea, we need an interpretation function that maps a flownomial expression to a boolean relation.

Let $\mathrm{Rel}(S)(m, n) = \{rel | rel \subseteq S^m \times S^n\}$ and $\mathrm{Rel}(S) = \mathrm{Rel}(S)(m, n)_{m,n \in IN}$. An *assignment* over a signature (A, F, r) is a mapping $\mathcal{I} : A \to \mathit{Rel}(\{0, 1\})$ such that for all $a \in A$ if $r(a) = (f, g)$ then $\mathcal{I}(a) \in \mathrm{Rel}(\{0, 1\})(|f|, |g|)$. The constants $\mathbf{I}_f$, $\wedge_2^f$ and ${}^{f_1}\times^{f_2}$, where $f, f_1, f_2 \in F^*$ are intepreted as follows: i) $\mathcal{I}(\mathbf{I}_f) = \{(x_1 \dots x_{|f|}, x_1 \dots x_{|f|}) \in \{0, 1\}^{|f|} \times \{0, 1\}^{|f|}\}$; ii) $\mathcal{I}(\wedge_2^f) = \{(x_1 \dots x_{|f|}, x_1 \dots x_{|f|} x_1 \dots x_{|f|}) \in \{0, 1\}^{|f|} \times \{0, 1\}^{2|f|}\}$; iii) $\mathcal{I}({}^{f_1}\times^{f_2}) = \{(x_1 \dots x_{|f_1|} y_1 \dots y_{|f_2|}, y_1 \dots y_{|f_2|} x_1 \dots x_{|f_1|}) \in \{0, 1\}^{|f_1|+|f_2|} \times \{0, 1\}^{|f_1|+|f_2|}\}$.

We can extend the interpretation function $\mathcal{I}$ over the set $\mathcal{E}(A, F, r)$. But first let us define some operations on relations in $\mathrm{Rel}(\{0, 1\})$, related to the data transformer model discussed in [4]. There is, however, one difference: the interpretation of the feedback operator. The operations are: i) if $r \in \mathrm{Rel}(\{0, 1\})(m, n)$ and $s \in \mathrm{Rel}(\{0, 1\})(p, q)$ then $rs \in \mathrm{Rel}(\{0, 1\})(m + p, n + q)$ is defined by $rs = \{(xz, yw) \,|\, (x, y) \in r \text{ and } (z, w) \in s\}$; ii) if $r \in \mathrm{Rel}(\{0, 1\})(m, n)$ and $s \in \mathrm{Rel}(\{0, 1\})(n, p)$ then $r \circ s \in \mathrm{Rel}(\{0, 1\})(m, p)$ is defined by $r \circ s = \{(x, z) \,|\, (x, y) \in r$ and $(y, z) \in s$ for some $y\}$ (note that this definition corresponds to the usual relation composition); iii) if $r \in \mathrm{Rel}(\{0, 1\})(m + p, n + p)$ then $r \uparrow^p = \{(x, y) \in \{0, 1\}^m \times \{0, 1\}^n \,|\, (x0 \dots 0, y0 \dots 0) \in r\}$.

The meaning of feedback given in [4] is related to steady state behavior in dynamic systems. We can interpret feedback flows as the state of our system and look for the input-output behaviors that correspond to steady states. This explains the equation used in [4] for defining the interpretation of the feedback operation. However, in our situation it is natural to ask for a steady state where no data is flowing through the network along the feedback flows. This explains our interpretation of the feedback operation.

We can now extend the interpretation function over the set $\mathcal{E}(A, F, r)$ of flownomial expressions: i) if $E_1, E_2 \in \mathcal{E}(A, F, r)$ then $\mathcal{I}(E_1 E_2) = \mathcal{I}(E_1)\mathcal{I}(E_2)$; ii) if $E_1, E_2 \in \mathcal{E}(A, F, r)$ and $E_1 \circ E_2$ is well defined, then $\mathcal{I}(E_1 \circ E_2) = \mathcal{I}(E_1) \circ \mathcal{I}(E_2)$; iii) if $E \in \mathcal{E}(A, F, r)$ and $E \uparrow^h$ is well defined, then $\mathcal{I}(E \uparrow^h) = \mathcal{I}(E) \uparrow^{|h|}$.

3.3 A Definition of Glass-Box Views

Glass-box views add constraints on the behavior of activities. Given the interpretation of an activity box as a boolean relation, it is natural to think of the glass-box view as a specification of the logic of selecting the inputs participating in the activity, and of the logic of generating the outputs produced by the activity. We can do this by assigning trees of connectors to the input and output of an activity box. We can easily describe these trees within the calculus of flownomials and then interpret them as sets of boolean strings. The cardinality of a tree T is defined as the number of its leaves and is denoted by $|T|$. Obviously, the cardinalities of the input and output trees must match the number of activity inputs and outputs.

Output trees and their interpretation are defined as: i) $\mathbf{I}_1$ is an output tree of cardinality 1 and $\mathcal{I}(\mathbf{I}_1) = \{1\}$; ii) if $T_1, \ldots, T_m$ are output trees and $m \geq 2$ then $\wedge_m^{and} \circ (T_1 \ldots T_m)$ is an output tree of cardinality $\sum_{i=1}^m |T_i|$ and $\mathcal{I}(\wedge_m^{and} \circ (T_1 \ldots T_m)) = \{(x_1 \ldots x_m) \,|\, x_i \in \mathcal{I}(T_i)\}$; iii) if $T_1, \ldots, T_m$ are output trees and $m \geq 2$ then $\wedge_m^{xor} \circ (T_1 \ldots T_m)$ is an output tree of cardinality $\sum_{i=1}^m |T_i|$ and $\mathcal{I}(\wedge_m^{xor} \circ (T_1 \ldots T_m)) = \{(x_1 \ldots x_m) \,|$ for some $i \in \{1, \ldots, m\}$, $x_i \in \mathcal{I}(T_i)$ and for all $j \neq i$, x_j is a string of $|T_j|$ zeros $\}$. If T is an output tree, then $\mathcal{I}(T)$ is the set of allowed output patterns at its leaves if an input is provided at its root.

Input trees and their interpretation are defined as: i) $\mathbf{I}_1$ is an input tree of cardinality 1 and $\mathcal{I}(\mathbf{I}_1) = \{1\}$; ii) if $T_1, \ldots, T_m$ are input trees and $m \geq 2$ then $\circ(T_1 \ldots T_m) \circ \vee_{and}^m$ is an input tree of cardinality $\sum_{i=1}^m |T_i|$ and $\mathcal{I}((T_1 \ldots T_m) \circ \vee_{and}^m) = \{(x_1 \ldots x_m) | x_i \in \mathcal{I}(T_i)\}$; iii) if $T_1, \ldots, T_m$ are input trees and $m \geq 2$ then $(T_1 \ldots T_m) \circ \vee_{or}^m$ is an input tree of cardinality $\sum_{i=1}^m |T_i|$ and $\mathcal{I}((T_1 \ldots T_m) \circ \vee_{or}^m) = \{(x_1 \ldots x_m)|$ for all $i \in \{1, \ldots, m\}$ either $x_i \in \mathcal{I}(T_i)\}$ or x_i is a string of $|T_i|$ zeros and there is $j \in \{1, \ldots, m\}$ such that $x_j \in \mathcal{I}(T_j)\}$. If T is an input tree, then $\mathcal{I}(T)$ is the set of input patterns that must be provided at its leaves to produce an output at its root.

Let T_I and T_O be the input and output trees in a glass-box view of an activity a. The pair (T_I, T_O) defines the boolean relation $p \subseteq \{0,1\}^{|T_I|} \times \{0,1\}^{|T_O|}$ as $p = (\mathcal{I}(T_I) \times \mathcal{I}(T_O)) \cup \{(0 \ldots 0, 0 \ldots 0)\}$. This means that if a pattern of inputs in $\mathcal{I}(T_I)$ is provided to activity a, then a pattern of outputs in $\mathcal{I}(T_O)$ can be produced. And for any boolean combination of length $|T_I|$ which is not in $\mathcal{I}(T_I)$, no output is produced. The behavior $(0 \ldots 0, 0 \ldots 0)$ added to p is called the *null behavior*. It models the fact that if no input is provided, then no output is produced.

For $x \in \{0,1\}^{|T_I|}$ let $p(x) = \{y | (x, y) \in p\}$ the image of x by p. $p(x)$ gives the set of all possible patterns of outputs that can be produced, given that the pattern x of inputs is provided. $p(x) = \emptyset$ means that no output can be produced because an undesired behavior occurs, such that an infinite loop without escape or a resource starvation. The interpretation we assign to activities and flownomial expressions via boolean relations is meant to rule out such undesired behaviors.

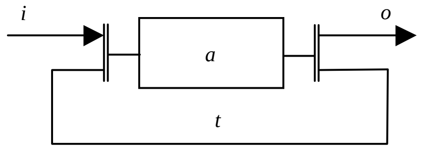

Figure 4. A looping process

For example, let us consider the process in figure 4. Obviously, if an input is available at i, then it can loop across flow t, but eventually it will leave activity a at output o. So the behavior $(1, 1)$ is allowed for the equivalent flownomial expression $a \uparrow^t$. However, if the input or connector is replaced with an and connector, path t will never be active, so activity a can never be executed, so no output is ever available at output o. The problem is that activity a is never activated because it suffers from *resource starvation* on input t, i.e it requires both i and t to be available, and only one is ever available, namely i. But if we replace the output exclusive or connector with an output and connector in figure 4, then path t will always be activated; in fact, our process enters an *infinite loop without escape*.

If an activity does not have a glass-box view, then it means that for any combination of inputs where at least one input is provided, any combination of outputs where at least one output is produced is allowed. In this situation it is natural to define p as $(\{0,1\}^m \setminus \{0 \ldots 0\}) \times ((\{0,1\}^n \setminus \{0 \ldots 0\}) \cup \{(0 \ldots 0, 0 \ldots 0)\})$.

4 Verifying Business Processes

Black-boxes in a business process model state a minimum consistency condition. They require that the flownomial expression corresponding to each activity decomposition to be well-formed, i.e. the signatures of the component subexpressions to match the requirements of the operations. This is insufficient if we take into account the glass-box views. They add additional constraints to the set of behaviors; the consistency conditions should be revised to account for this.

Let $a \in A$ be an activity and let E be the flownomial expression of its decomposition. To each activity that occurs in E we associate an interpretation according to those stated in subsection 3.3. Then we can easily compute the interpretation of E by applying the rules from subsection 3.2. In this way we obtain the set of possible behaviors of the decomposition of a.

We can say that the decomposition of a is *inconsistent* by examining solely its interpretation $\mathcal{I}(E)$. E is called inconsistent if the only possible behavior is the null behavior, i.e. $\mathcal{I}(E) = \{(0\ldots0, 0\ldots0)\}$. Obviously, we are interested in consistent decompositions. An inconsistent decomposition suggests that the corresponding subprocess is problematic because no behavior different from the null one is allowed.

Also, by examining the set $\mathcal{I}(E)$ we can identify those patterns of inputs which provided to the decomposition of a might cause it to enter either an infinite loop without escape or a resource starvation.

For example consider the process in figure 2, with the glass-box views in figure 3. To avoid long names, we shall use the notations from subsection 3.1. According to the rules in subsection 3.3, the interpretations of a_1, a_2, a_3, a_4 are $\mathcal{I}(a_1) = \{(0,0),(1,1)\}$, $\mathcal{I}(a_2) = \{(00,00),(01,10),(01,01),(10,10),(10,01),(11,10),(11,01)\}$, $\mathcal{I}(a_3) = \{(0,0),(1,1)\}$, $\mathcal{I}(a_4) = \{(00,0),(11,1)\}$. Applying the rules from subsection 3.1 we get $\mathcal{I}(E) = \{(00,00),(11,11)\}$, where E is the flownomial expression introduced in subsection 3.1.

Note that $(10,10) \notin \mathcal{I}(E)$. This violates our intuition about the process, i.e. when only the *Material Request* is provided we'd expect it to be able to produce only a *Validated Request*. Because $(10,10) \notin \mathcal{I}(E)$, this is impossible within our model. The conclusion is that the model is incorrect. A closer look at figure 3 shows that if only input *Material Request* is present, then our process suffers from resource starvation on input *New Supplier Requirement* at activity *Develop New Supplier Spec*. The cause is the flow *Validated Request* that forks to the output of the process and to the input of *Develop New Supplier Spec*. One solution is to replace the fork with an one input/two output activity of arity (o_1, o_1o_1) that allows the input to be passed also only to the first of its outputs.

5 Conclusions

By writing black-box diagrams as flownomial expressions and interpreting them over boolean relations, we can assign them a meaning. Moreover, by attaching glass-view boxes to the black-boxes, we can constraint the behaviors of the model. We thus obtained an IDEF-based notation of business process models that is both easy to use and formal. Its main advantages are: i) it is based on the well known boxes, arrows and connectors employed by IDEF0 and IDEF3, which are widely used in practice and ii) because it has a meaning underpinned by maths makes the models suitable to formal analysis and validation. We intend to use these results in a module within the INSPIRE tool to verify business processes developed with this notation.

References

[1] M. Ould, *Business Processes: Modelling and Analysis for Reengineering and Improvement*, John Wiley & Sons, 1995.

[2] Draft Federal Information Processing Standards Publication 183, *Integration Definition for Function Modelling (IDEF0)*, 1993

[3] R. Mayer et al., *Information Integration for Concurrent Engineering (IICE): IDEF3 Process Description Capture Method Report*, 1993

[4] G. Ştefănescu, *Network Algebras*, Springer-Verlag, 2000

[5] W. Janssen, R. Mateescu, S. Mauw, P. Fennema and P. van der Stappen, Model Checking for Mangers, *Theoretical and Practical Aspects of SPIN model checking: 5th and 6th International SPIN Workshops*, LNCS 1680, 1999, 92-107

[6] M. Schroeder, Verification of Business Processes for a Correspondence Handling Center Using CCS, *Proc. of European Symposium on Validation and Verification of Knowledge Based Systems and Components*, Oslo, Norway, 1999, 253-264

[7] E. Yu, J. Mylopoulos, Y. Lesprance, AI Models for Business Process Reengineering, *IEEE Expert*, 11(4), 1996, 16-23

[8] W.M.P. van der Aalst and A.H.M. ter Hofstede, Verification of Workflow Task Structures: A Petri-net-based Approach, *Information Systems*, 25(1), 2000, 43-69

[9] V. Thurner, A Formally Founded Description for Business Processes. *Proc.PDSE-98: International Symposium on Software Engineering for Parallel and Distributed Systems*, Los Alamitos, California, 1998, 254-261

[10] B. Rumpe, V. Thurner, Refining Business Processes, *Proc.7th OOPSLA Workshop on Behavioral Semantics of OO Business and System Specifications*, Munich, Germany, 1998, 205-220

[11] C. Fox, The Process Representation Module (Specification), INSPIRE (IST-1999-10387) Deliverable 2.1, 2000